UNIVERSIDAD DEL VALLE DE GUATEMALA

Laboratorio 2 pt1

Andre Marroquin Tarot - 22266
Sergio Orellana - 221122

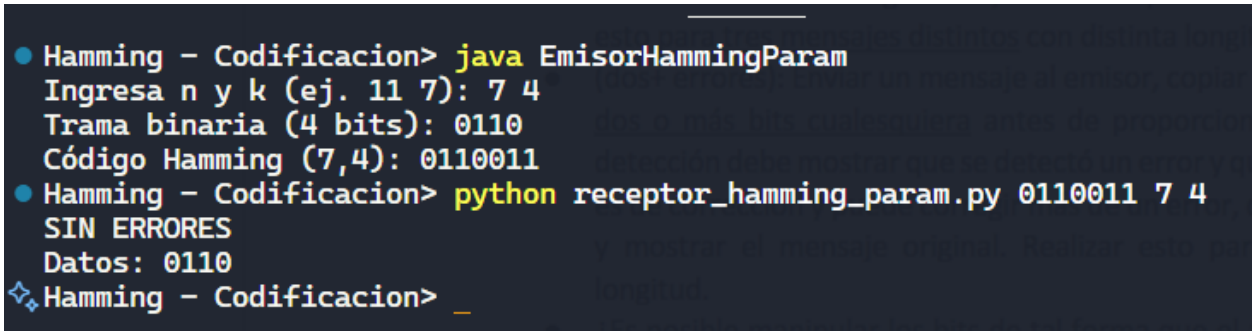
Redes

Link Git https://github.com/mar22266/Redes-L2.git

Pruebas

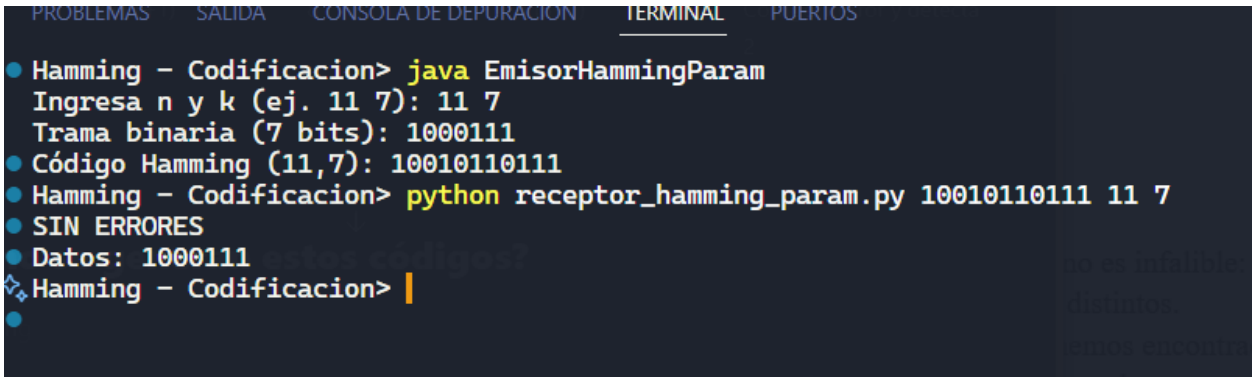
Sin errores

1.



Trama: 0110
Mensaje emisor: 0110011
Mensaje receptor: 0110011

2.



Trama: 1000111
Mensaje emisor: 10010110111
Mensaje receptor: 10010110111

3.

```
Hamming - Codificacion> java EmisorHammingParam
Ingresa n y k (ej. 11 7): 15 11
Trama binaria (11 bits): 11100010110
Código Hamming (15,11): 111000100111010
Hamming - Codificacion> python receptor_hamming_param.py 111000100111010 15 11
SIN ERRORES
Datos: 11100010110
Hamming - Codificacion>
```

Trama: 11100010110
Mensaje emisor: 111000100111010
Mensaje receptor: 111000100111010

Con un error:

1.

```
Hamming - Codificacion> java EmisorHammingParam
Ingresa n y k (ej. 11 7): 8 4
Trama binaria (4 bits): 0110
Código Hamming (8,4): 00110011
Hamming - Codificacion> python receptor_hamming_param.py 00110111 8 4
ERROR CORREGIDO en posición 3
Datos: 0110
Hamming - Codificacion>
```

Trama: 0110
Mensaje emisor: 00110011
Mensaje receptor: 00110111

2.

```
Hamming - Codificacion> java EmisorHammingParam
Ingresa n y k (ej. 7 4): 7 4
Mensaje binario (cualquier longitud): 101011

Padding añadido (ceros): 2
Palabras Hamming (7,4) concatenadas:
10100101100001
Hamming - Codificacion> python receptor_hamming_param.py 10100101101001 7 4 2
Bloque 1: SIN ERRORES  → datos 1010
Bloque 2: CORREGIDO bit 4  → datos 1100

Mensaje reconstruido: 101011
Hamming - Codificacion>
```

Trama: 101011
Mensaje emisor: 10100101100001
Mensaje receptor: 10100101101001

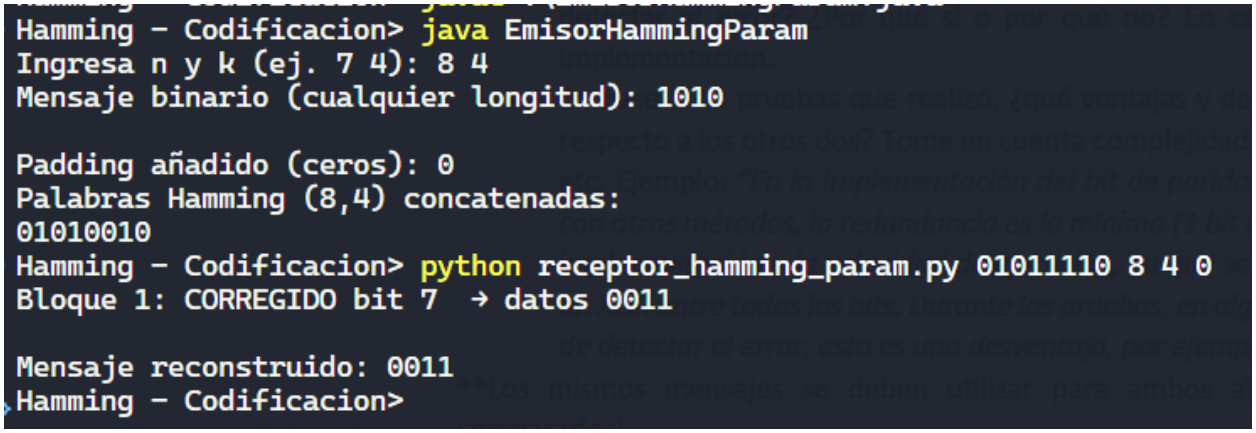
3.

```
Hamming - Codificacion> java EmisorHammingParam
Ingresa n y k (ej. 11 7): 15 11
Trama binaria (11 bits): 10101010101
Código Hamming (15,11): 101010100101101
Hamming - Codificacion> python receptor_hamming_param.py 101010000101101 15 11
ERROR CORREGIDO en posición 9
Datos: 10101010101
Hamming - Codificacion>
```

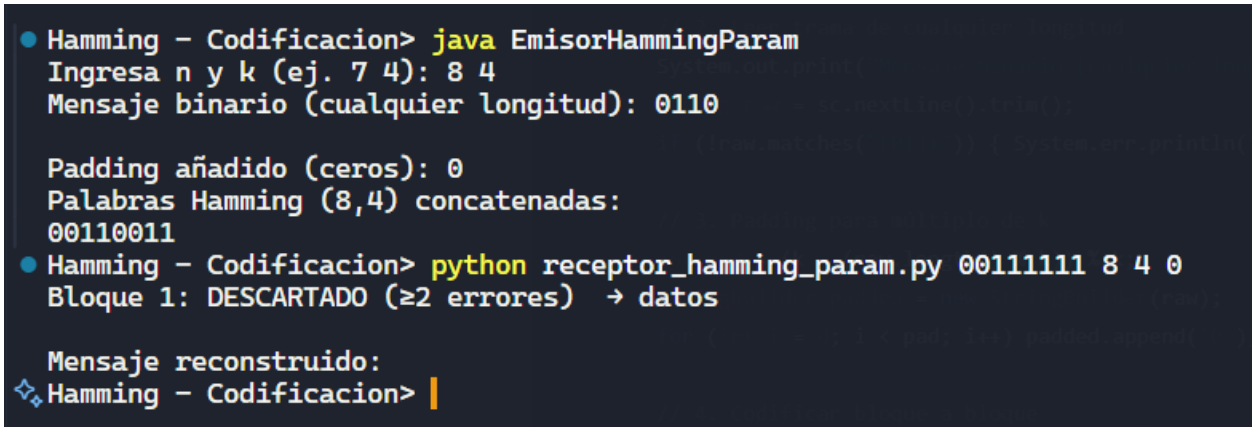
Trama: 10101010101
Mensaje emisor: 101010100101101
Mensaje receptor: 101010**0**00101101

Con dos o más errores:

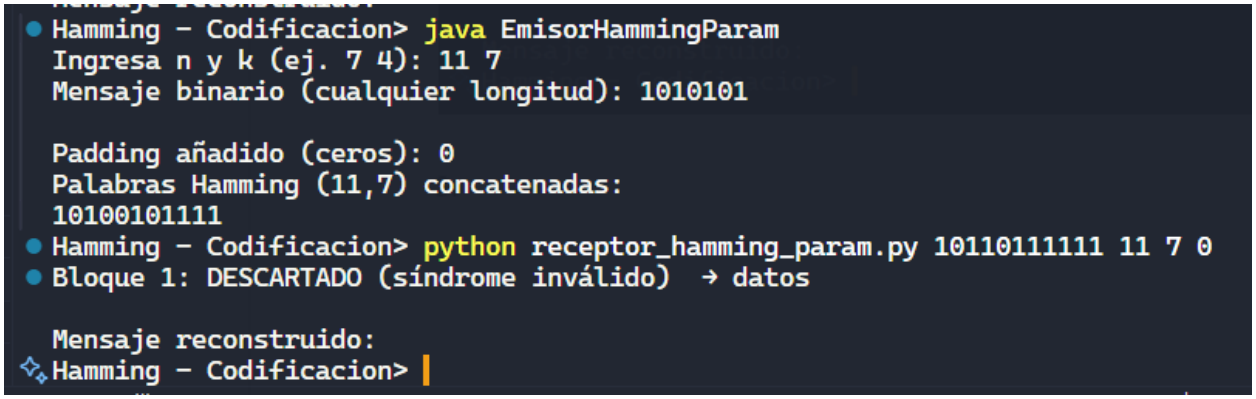
1.



2.



3.



4.

```
Hamming - Codificacion> java EmisorHammingParam
Ingresa n y k (ej. 7 4): 7 4
Mensaje binario (cualquier longitud): 101011

Padding añadido (ceros): 2
Palabras Hamming (7,4) concatenadas:
10100101100001
Hamming - Codificacion> python receptor_hamming_param.py 10111101101101 7 4 2
Bloque 1: CORREGIDO bit 7  → datos 0011
Bloque 2: CORREGIDO bit 7  → datos 0101

Mensaje reconstruido: 001101
Hamming - Codificacion>
```

Cuando trabajo con Hamming (7, 4) me limito a un esquema que sólo puede ubicar y corregir un bit erróneo; si ocurren dos, existe la posibilidad de que su combinación produzca un síndrome indistinguible del de un único error y el algoritmo se confunda. Por eso algunas tramas con doble fallo parecen "arreglarse" cuando en realidad quedan mal: la ambigüedad es inherente al código, no a la programación.

Para mejorar esa debilidad añado la paridad global y paso a Hamming extendido (8, 4). Con ese octavo bit, el algoritmo verifica si el número total de unos recibidos es par o impar; así logra detectar –aunque no reparar– cualquier par de errores. Sin embargo, el alcance sigue siendo limitado: con tres o más fallos la paridad vuelve a coincidir con la original y el patrón resultante puede confundirse con un solo error, de modo que el decodificador intentará corregir donde no debe. En otras palabras, el (8, 4) amplía la detección a dos errores pero sigue sin garantizar fiabilidad absoluta cuando la alteración afecta a tres o más bits, razón por la cual a veces "funciona" y otras no, dependiendo del tipo exacto de corrupción que sufra la palabra.

¿Puede evadirse la detección manipulando bits?

Sí, y lo comprobé con mi propia implementación. Tomé la palabra Hamming (7, 4) que el emisor genera para los datos 1100: el código sale 1100001. Luego cambié dos posiciones (pasé a 1101101). Cuando ejecuto el receptor, el programa indica "CORREGIDO bit 7" y me devuelve los datos 0101. En realidad introduje dos errores (bits 4 y 5), pero el síndrome resultante coincide con el patrón de un único error en la posición 7; el algoritmo se confunde porque la distancia mínima del (7, 4) es 3 y no puede distinguir ciertas parejas de errores. Hice lo mismo con Hamming (8, 4): a la palabra correcta 01100001 le cambié los mismos dos bits (01101101). El receptor extendido, que debería descartar al detectar dos fallos, vuelve a decir "CORREGIDO bit 7". El motivo es que aquí la paridad global R8 también quedó alterada (pasa

de 1 a 1), de modo que el esquema SEC-DED interpreta la situación como un único error. Concluí que, aunque (8, 4) detecta muchas combinaciones dobles, todavía existen patrones específicos donde la paridad global no discrimina y el algoritmo se equivoca.

Conclusiones:

- Determiné que el bit de paridad añade un único bit y basta un XOR global para generarlo, por lo que su implementación es muy simple y veloz; sin embargo, no localiza errores y cualquier cambio que modifique un número par de bits pasa inadvertido.
- Verifiqué que el Hamming (7, 4) puede corregir un solo error y suele advertir la presencia de dos, pero existen pares de bits cuyo síndrome imita un error único, de modo que el algoritmo "corrige" el bit equivocado y entrega datos alterados.
- Comprobé que el Hamming extendido (8, 4) emplea un bit de paridad global para detectar la mayoría de los pares de errores y descartar la palabra; aun así, si el canal introduce tres o más fallos en ciertas posiciones, el esquema puede confundirlos con un error único y actuar de forma incorrecta.
- Observé que el Hamming (11, 7) reparte mejor la redundancia al proteger siete bits de datos con los mismos cuatro bits de control; corrige un error igual que el (7, 4), pero al no incorporar paridad global aún existe la posibilidad de que dos errores específicos queden sin detectar.
- Constaté que el Hamming (15, 11) ofrece la mayor eficiencia de datos frente a bits de control y mantiene la corrección de un error; no obstante, la longitud mayor incrementa la probabilidad de que varios fallos coincidan en la misma palabra y, al carecer de paridad global, algunos pares de errores siguen pudiendo camuflarse.

Referencias:

- AIX. (2025). https://www.ibm.com/docs/es/aix/7.2.0?topic=parameters-parity-bits

- 7.2.4.1.3 Fletcher Algorithms. (2025). (C) Copyright 2025. https://onlinedocs.microchip.com/oxy/GUID-4DC87671-9D8E-428A-ADFE-98D694F9F089-en-US-4/GUID-7399BDB2-CC51-4E98-AE8A-90E48F9167DF.html
-
- Profundizacion en CRC32 un mecanismo de deteccion de errores mas solido - FasterCapital. (2024). FasterCapital. https://fastercapital.com/es/contenido/Profundizacion-en-CRC32--un-mecanismo-de-deteccion-de-errores-mas-solido.html
- TutorialsPoint. (2020, 27 junio). Hamming code for single error correction, double error detection.

https://www.tutorialspoint.com/hamming-code-for-single-error-correction-double-error-detection

- GeeksforGeeks. (2025, 14 mayo). Hamming Code in Computer Network. GeeksforGeeks. https://www.geeksforgeeks.org/computer-networks/hamming-code-in-computer-network/